Large-scale Conic Optimization: A Neural Fixed Point Acceleration Approach

Pan Bikang
ShanghaiTech University
2019533222
panbk@shanghaitech.edu.cn

Gu Sizhe
ShanghaiTech University
2019533200
guszh@shanghaitech.edu.cn

Gou Boyu
ShanghaiTech University
2019533179
gouby@shanghaitech.edu.cn

Abstract

SCS is the state-of-art solver for large scale convex cone optimization. We apply neural fixed-point acceleration which combines ideas from meta-learning and classical acceleration methods to automatically learn to accelerate fixed-point problem in SCS solver. Our work verifies the performance of neural acceleration in several convex optimization problems, including Lasso, Robust PCA, Ridge, Compressed Sensing and the SOCP problem in dense wireless cooperative networks. The source code behind this paper is available at github-link

1 Introduction

In this paper we apply *neural fixed-point acceleration* Venkataraman and Amos [2021] to a splitting conic solver for large scale conic optimization.

Recall the operator splitting method – *alternating direction method of multipliers*(ADMM) proposed by O'donoghue *et al.* [2016], we can treat this algorithm as a map $f : R^n \to R^n$ and the iterations repeatedly apply f until the solution reach and provably converge under assumptions of f. In other words, iteration converge also means the update reaches the fixed-point: $x = f(x)$. Therefore, we can apply the neural fixed-point method to accelerate the iteration process.

This approach has several favorable properties. In fixed-point iterative algorithms, classic solvers can often be done without learning. For many real-time applications, though, traditional fixed-point solvers can be too slow. Instead, we introduces a faster, but lower-accuracy, solution to meet the needs of these real world applications. However, learning to optimize and acceleration are notoriously hard problems with instabilities and poor solutions. Through careful design of models and loss functions, we address the challenges of learning over unrolled SCS computations and finally apply this method to Robust PCA, the SOCP problem for *dense wireless cooperative networks* proposed by Shi *et al.* [2015] and many other (convex) optimize problems.

1.1 Related work

Learning to Optimize and meta-Learning The machine learning community has recently explored many approaches to learning to improve the solutions to optimization problems. These applications have wide-ranging applications, e.g. in optimal power flow Baker [2020], combinatorial optimization Khalil *et al.* [2016]; Dai *et al.* [2017]; Nair *et al.* [2020]; Bengio *et al.* [2021], and differential equations Li *et al.* [2020];Poli *et al.* [2020];Kochkov *et al.* [2021]). The meta-learning and learning to optimize literature, e.g. (Li and Malik [2016]; Finn *et al.* [2017]; Wichrowska *et al.* [2017]; Andrychowicz *et al.* [2016]; Metz *et al.* [2019]Metz *et al.* [2021]; Gregor and LeCun [2010]), focuses on learning better solutions to parameter learning problems that arise for machine learning tasks. Bastianello *et al.* [2021] approximates the fixed-point iteration with the closest contractive fixed-point iteration.

Fixed-point problems and acceleration. Accelerating fixed-point computations date back decades and include Anderson Acceleration (AA) (Anderson [1965]) and Broyden's method (Broyden [1965]), or variations such as Walker and Ni [2011]; Zhang *et al.* [2020]. Traditional SCS allows AA but it may cause instability to the result.

2 Problem Formulation

2.1 SCS model

First consider the primal-dual pair of (convex) cone optimization problems

$$\begin{array}{lll} \min & c^T x & \max -b^T y \\ \text{s.t.} & Ax + s = b & \text{s.t.} -A^T y + r = c \\ & (x, s) \in \mathbb{R}^n \times \mathcal{K} & (r, y) \in \{0\}^n \times \mathcal{K}^* \end{array} \tag{1}$$

Here $x \in \mathbb{R}^n$ and $s \in \mathbb{R}^m$ (with $n \leq m$) are the primal variables, and $r \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ are the dual variables. We refer to x as the primal variable, s as the primal slack variable, y as the dual variable, and r as the dual residual. The set $\mathcal{K}$ is a nonempty, closed, convex cone with dual cone $\mathcal{K}^*$, and $\{0\}^n$ is the dual cone of $\mathbb{R}^n$, so the cones $\mathbb{R}^n \times \mathcal{K}$ and $\{0\}^n \times \mathcal{K}^*$ are duals of each other. The problem data are $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and the cone $\mathcal{K}$. (We consider all vectors to be column vectors).

2.1.1 Self-dual embedding

The original pair of problems (1) can be converted into a single feasibility problem by embedding the KKT conditions into a single system of equations and inclusions that the primal and dual optimal points must jointly satisfy. The embedding is as follows:

$$\begin{bmatrix} r \\ s \\ \kappa \end{bmatrix} = \begin{bmatrix} 0 & A^T & c \\ -A & 0 & b \\ -c^T & -b^T & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \tau \end{bmatrix}, (x, s, r, y, \tau, \kappa) \in \mathbb{R}^n \times \mathcal{K} \times \{0\}^n \times \mathcal{K}^* \times \mathbb{R}_+ \times \mathbb{R}_+ \tag{2}$$

To simplify the notation, let

$$u = \begin{bmatrix} x \\ y \\ \tau \end{bmatrix}, v = \begin{bmatrix} r \\ s \\ \kappa \end{bmatrix}, Q = \begin{bmatrix} 0 & A^T & c \\ -A & 0 & b \\ -c^T & -b^T & 0 \end{bmatrix} \tag{3}$$

The homogeneous self-dual embedding can then be expressed as

$$\begin{array}{rl} \text{find} & (u, v) \\ \text{s.t.} & v = Qu \\ & (u, v) \in \mathcal{C} \times \mathcal{C}^* \end{array} \tag{4}$$

where $\mathcal{C} = \mathbb{R}^n \times \mathcal{K}^* \times \mathbb{R}_+$ is a cone with dual cone $C^* = \{0\}^n \times \mathcal{K} \times R_+$ The resultant homogeneous self-dual embedding is further solved via the operator splitting method, a.k.a. the ADMM algorithm.

2.1.2 ADMM Iteration

To apply ADMM, we transform the embedding (4) to ADMM form:

$$\text{minimize}[I_{\mathcal{C}\times\mathcal{C}^*}(u, v) + I_{Qu=v}(\tilde{u}, \tilde{v})]\text{s.t.}(u, v) = (\tilde{u}, \tilde{v}) \tag{5}$$

where I_S is the indicator function of the set S,i.e., $I_S(z)$ is zero for $z \in S$ and it is $+\infty$ otherwise.

Applying the ADMM algorithm to the problem and eliminating the dual variables by exploiting the self-dual property of the problem (4), the final algorithm is shown as follows:

$$\begin{aligned} \tilde{u}^{k+1} &= (I+Q)^{-1}(u^k + v^k) \\ u^{k+1} &= \Pi_{\mathcal{C}}(\tilde{u}^{k+1} - v^k) \\ v^{k+1} &= v^k - \tilde{u}^{k+1} + u^{k+1} \end{aligned} \tag{6}$$

where $\Pi_{\mathcal{C}}(x)$ denotes the Euclidean projection of x onto the set $\mathcal{C}$.

2.2 Neural fixed-point Model for SCS

Algorithm 1 Neural fixed-point acceleration augments standard fixed-point computations with a learned initialization and updates to the iterates.

```
Input: Context $\phi$, parameters $\theta$, and fixed-point map $f$
$[x_1, h_1] = g_\theta^{init}(\phi)$                          ▷ Initial hidden state and iterate
for fixed-point iteration $t = 1..T$ do
$\tilde{x}_{t+1} = f(x_t; \phi)$                               ▷ Original fixed-point iteration
$x_{t+1}, h_{t+1} = g_\theta^{acc}(x_t, \tilde{x}_{t+1}, h_t)$              ▷ Acceleration
end for
```

We are interested in settings and systems that involve solving a known distribution over fixed-point problems. Each fixed-point problem depends on a $context$ $\phi \in \mathbb{R}^m$ that we have a distribution over $\mathcal{P}(\phi)$. The distribution $\mathcal{P}(\phi)$ induces a distribution over fixed-point problems $f(x; \phi) = x$ with a fixed-point map f that depends on the context. Informally, our objective will be to solve this class of fixed-point problems as fast as possible. Notationally, other settings refer to ϕ as a "parameter" or "conditioning variable," but here we will consistently use "context." We next consider a general solver for fixed-point problems that captures classical acceleration methods as an instance, and can also be parameterized with some and learned to go beyond classical solvers. Given a fixed context ϕ, we solve the fixed-point problem with Alg. 1. At each time step t we maintain the *fixed-point iterations* x_t and a *hidden state* h_t. The initializer g_θ^{init} depends on the context ϕ. It provides the starting iterate and hidden state. The acceleration g_θ^{acc} updates the iterate after observing the application of the fixed-point map f.

2.2.1 Modeling and optimization

We first parameterize the models behind the fixed-point updates in Alg. 1. In neural acceleration, we will use learned models for g_θ^{init} and g_θ^{acc} . We experimentally found that we achieve good results a standard MLP for g_θ^{init} and a recurrent model such as an LSTM (Hochreiter and Schmidhuber [1997]) or GRU (Cho *et al.* [2014]) for g_θ^{acc} . While the appropriate models vary by application, a recurrent structure is a particularly good fit as it encapsulates the history of iterates in the hidden state, and uses that to predict a future iterate.

Next, we define and optimize an objective for learning that characterizes how well the fixed-point iterations are solved. Here, we use the *fixed-point residual norms* defined by $\mathcal{R}(x; \phi) := ||x - f(x; \phi)||_2$. This is a natural choice for the objective as the convergence analysis of classical acceleration methods are built around the fixed-point residual. Our learning objective is thus to find the parameters to minimize the fixed-point residual norms in every iteration across the distribution of fixed-point problem instances, i.e.

$$\min_\theta \quad \mathbb{E}_{\phi \sim \mathcal{P}(\phi)} \sum_{t<T} \mathcal{R}(x_t; \phi) / \mathcal{R}_0(\phi) \tag{7}$$

where T is the maximum number of iterations to apply and $\mathcal{R}_0$ is a normalization factor that is useful when the fixed-point residuals have different magnitudes. We optimize (7) with gradient descent, which requires the derivatives of the fixed-point map $\nabla_x f(x)$.

2.2.2 Designing Neural SCS

We now describe how we design Neural SCS as a realization of Alg. 1 in three key steps: modeling, differentiating through SCS, and designing the objective.

Modeling. The input parameters θ come from the initialization of the neural networks that we train, g_θ^{init} init and g_θ^{acc} . To construct the input context ϕ for a problem instance, we convert the problem instance into its standard form (1), and use the quantities A, b and c, i.e. $\phi = [v(A); b; c]$ where $v : \mathbb{R}^{m\times n} \to \mathbb{R}^{mn}$ vectorizes the matrix A. We use an MLP for g_θ^{init}, and a multi-layer LSTM or GRU for g_θ^{acc}.

Differentiating through SCS. Optimizing the loss in (7) requires that we differentiate through the fixed-point iterations of SCS: 1) For the *linear system solve*. We use implicit differentiation, e.g. as described in Barron and Poole [2016]. Further, for differentiating through SCS, for a linear system $Qu = v$, we only need to obtain the derivative $\frac{\partial u}{\partial v}$, since the fixed-point computation repeatedly solves linear systems with the same Q, but different v. This also lets us use an LU decomposition of Q to speed up the computation of the original linear system solve and its derivative. 2) for the *cone projections*, we use the derivatives from Ali *et al.* [2017]; Busseti *et al.* [2019].

Designing the Loss. The natural choice for the learning objective is the fixed-point residual norm of SCS. With this objective, the interacting algorithmic components of SCS cause g_θ^{acc} and g_θ^{init} to learn poor models for the cone problem. In particular, SCS scales the iterates of feasible problems by τ for better conditioning. However, this causes a serious issue when optimizing the fixed-point residuals: shrinking the iterate-scaling τ artificially decreases the fixed-point residuals, allowing g_θ^{acc} to have a good loss even with poor solutions.

We eliminate this issue by normalizing each x_t by its corresponding τ, similar to Busseti *et al.* [2019]. Thus, the fixed-point residual norm becomes the $||x_t/\tau_t - f(x_t,\phi)/\tau_{f(x_t,\phi)}||$. We are then always measuring the residual norm with τ = 1 for the learning objective, which does not modify the cone program that we are optimizing In addition, with this objective, we no longer need to learn or predict from τ in the models g_θ^{init} and g_θ^{acc} .

3 Application

3.1 Modelling

In this section, we provide problem models to deal with SCS+Neural. We will introduce the different cone problems we are going to handle at first, and then some additional experimental setup details.

3.1.1 Lasso

Lasso (Tibshirani [1996]), is a well-known machine learning problem formulated as follows:

$$\min_z (1/2)\|Fz - g\|_2^2 + \mu\|z\|_1 \tag{8}$$

where $z \in \mathbb{R}^p$, $F \in \mathbb{R}^{q\times p}$, $g \in \mathbb{R}^p$ and $\mu \in \mathbb{R}_+$ are data. In our experiments, we draw problem instances from the same distributions as O'donoghue *et al.* [2016] and Venkataraman and Amos [2021], generate F as $q \times p$ matrix with entries from $\mathcal{N}(0,1)$, then generate a sparse vector z^* with entries from $\mathcal{N}(0,1)$, and set a random 90% of its entries to 0. Finally, we compute $g = Fz^* + w$, where $w \sim \mathcal{N}(0, 0.1)$. We set $\mu = 0.1\|F^T g\|_\infty$. We use $p = 100$ and $q = 50$.

3.1.2 Ridge

Ridge (Hoerl and Kennard [1970]), is also a machine learning problem formulated as follows:

$$\min_z (1/2)\|Fz - g\|_2^2 + \mu\|z\|_2^2 \tag{9}$$

where $z \in \mathbb{R}^p$, and where $F \in \mathbb{R}^{q\times p}$, $g \in \mathbb{R}^p$ and $\mu \in \mathbb{R}_+$ are data. In our experiments, Similar to lasso, we draw problem instances from the same distributions as O'donoghue *et al.* [2016] and Venkataraman and Amos [2021], generate F as $q \times p$ matrix with entries from $\mathcal{N}(0,1)$, then generate a sparse vector z^* with entries from $\mathcal{N}(0,1)$, and set a random 90% of its entries to 0. Finally, we compute $g = Fz^* + w$, where $w \sim \mathcal{N}(0, 0.1)$. We set $\mu = 0.1\|F^T g\|_\infty$. We use $p = 100$ and $q = 50$.

3.1.3 Robust PCA

Robust Principal Components Analysis (Candès *et al.* [2011]) aims to recover a low rank matrix from highly corrupted measurements by solving:

$$\begin{array}{ll} \min & \|L\|_* \\ \text{s.t.} & \|S\|_1 \leq \mu \\ & L + S = M \end{array} \tag{10}$$

where variable $L \in \mathbb{R}^{p\times q}$ is the original low-rank matrix, variable $S \in \mathbb{R}^{p\times q}$ is the noise matrix, the data is $M \in \mathbb{R}^{p\times q}$ the matrix of measurements, and $\mu \in \mathbb{R}_+$ is the constant that constrains the corrupting noise term.

Again, we draw problem instances from the same distributions as O'donoghue *et al.* [2016]: we generate a random rank-r matrix L^*, and a random sparse matrix S^* with no more than 10% non-zero entries. We set $\mu = \|S^*\|_1$, and $M = L^* + S^*$. We use $p = 30$, $q = 3$ and $r = 2$.

3.1.4 SOCP problem in dense wireless cooperative networks

The **signal model** founded by Shi *et al.* [2015] considers a dense fully cooperative network with L RAUs(radio access unit) and K single-antenna MUs(mobile user), where the l-th RAU is equipped with N_l antennas. The centralized signal processing is performed at a central processor and the propagation channel from the l-th RAU to the k-th MU is denoted as $h_{kl} \in \mathbb{C}^{N_l}, \forall k, l$. We focus on the downlink transmission for illustrative purpose. But our proposed approach can also be applied in the uplink transmission, as we only need to exploit the convexity of the resulting performance optimization problems. The received signal $y_k \in \mathbb{C}$ at MU is given by

$$y_k = \sum_{l=1}^{L} h_{kl}^H v_{lk} s_k + \sum_{i\neq k} \sum_{l=1}^{L} h_{kl}^H v_{lk} s_i + n_k, \forall k, \tag{11}$$

where s_k is the encoded information symbol for MU k with $\mathbb{E}[|s_k|^2] = 1$, $v_{lk} \in \mathbb{C}^{N_l}$ is the transmit beamforming vector from the l-th RAU to the k-th MU, and $n_k \sim \mathcal{CN}(0, \sigma_k^2)$ is the additive Gaussian noise at MU k. We assume that s_k's and n_k's are mutually independent and all the users apply single user detection. Thus the signal-to-interference-plus-noise ratio (SINR) of MU k is given by

$$\Gamma_k(v) = \frac{|h_k^H v_k|^2}{\sum_{i\neq k} |h_k^H v_i|^2 + \sigma_k^2}, \forall k, \tag{12}$$

where $h_k \triangleq [h_{k1}^T, ..., h_{kL}^T]^T \in \mathbb{C}^N$ with $N = \sum_{l=1}^{L} N_l, v_k \triangleq [v_{1k}^T, ..., v_{Lk}^T]^T \in \mathbb{C}^N$ and $v \triangleq [v_1^T, ..., v_K^T] \in \mathbb{C}^{NK}$. We assume that each RAU has its own power constraint,

$$\sum_{k=1}^{K} \|v_{lk}\|_2^2 \leq P_l, \forall l, \tag{13}$$

where $P_l > 0$ is the maximum transmit power of the l-th RAU. In this paper, we assume that the full and perfect CSI(channel state information) is available at the central processor and all RAUs only provide unicast/broadcast services.

The following **channel model** for the link between the k-th MU and the l-th RAU is:

$$h_{kl} = 10^{-L(d_{kl})/20} \sqrt{\varphi_{kl} s_{kl}} f_{kl}, \forall k, l \tag{14}$$

where $L(d_{kl})$ is the path-loss in dB at d_{kl} distance, s_{kl} is the shadowing coefficient, φ_{kl} is the antenna gain and f_{kl} is the small-scale fading coefficient.

Finally, with the preliminaries and definitions before, we can write the problem $\mathcal{P}_{cvx}$ (Grant and Boyd [2008]):

$$\begin{array}{ll} \min & \|v\|_2 \\ \text{s.t.} & \|D_l v\|_2 \leq \sqrt{P_l}, l = 1, ..., L \\ & \|C_k v + g_k\|_2 \leq \beta_k r_k^T v, k = 1, ..., K, \end{array} \tag{15}$$

where $D_l = \textbf{blkdiag}\{D_l^1, ..., D_l^K\} \in \mathbb{R}^{N_l K \times NK}$ with $D_l^k = [0_{N_l \times \sum_{i=1}^{l-1} N_i}, I_{N_l \times N_l}, 0_{N_l \times \sum_{i=l+1}^{l-1} N_i}] \in \mathbb{R}^{N_l \times N}$, $\beta_k = \sqrt{1 + 1/\gamma_k}$, $r_k = [0^T_{(k-1)N}, h_k^T, 0^T_{(K-k)N}]^T \in \mathbb{R}^{NK}$, $g_k = [0_K^T, \sigma_k]^T \in \mathbb{R}^{K+1}$, and $C_k = [\widetilde{C}_k, 0_{NK}]^T \in \mathbb{R}^{(K+1)\times NK}$ with $\widetilde{C}_k = blkdiag\{h_k, ..., h_k\} \in \mathbb{R}^{NK \times K}$.

3.1.5 Compressed sensing

Compressed sensing Donoho [2006], also called compressive sampling or sparse sampling, is a signal processing technique for efficiently acquiring and reconstructing a signal. The form of the model is as follows,

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \|\boldsymbol{x}\|_1 \\ \text{subject to} \quad & \boldsymbol{Ax} = \boldsymbol{z}, \end{aligned} \tag{16}$$

Where $x \in \mathbb{R}^m$ is the original sparse signal and $A \in \mathbb{R}^{n \times m}$ is the transforming matrix. We want to recover x from z with less noise. We use $m = 100, n = 20$ and density is 0.2.

4 Experiments

4.1 Re-implement the essay

We first re-implemented Lasso (Tibshirani [1996]) and Robust PCA (Candès *et al.* [2011]) problem experimented in the essay *Neural Fixed-Point Acceleration for Convex Optimization* Venkataraman and Amos [2021]. Similar to the essay, our experimental results also focus on the number of iterations required to achieve required accuracy with SCS+Neural and we are going to compare SCS, SCS+AA, and SCS+Neural under this standard. The results for Lasso are shown in Fig. 1 and Fig. 2.

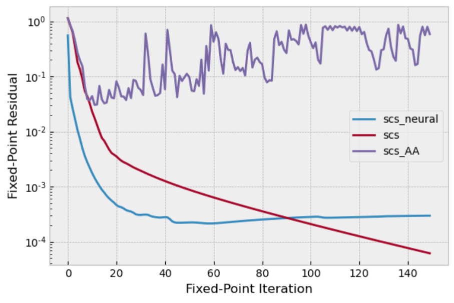

Figure 1: SCS+LSTM for Lasso

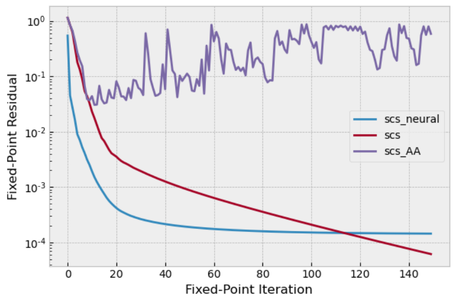

Figure 2: SCS+GRU for Lasso

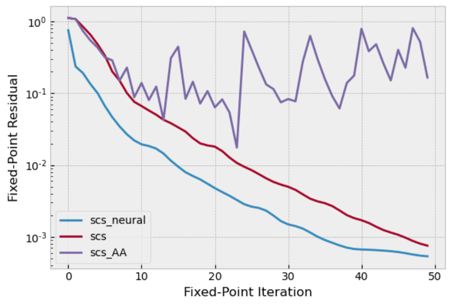

Figure 3: SCS+LSTM for Robust PCA

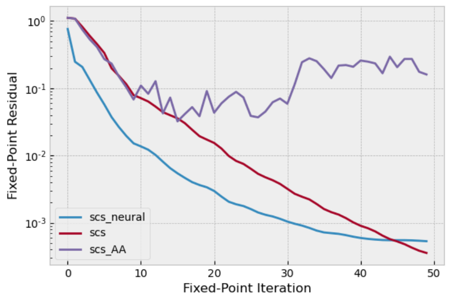

Figure 4: SCS+GRU for Robust PCA

As is shown above, SCS+Neural reaches a low residual faster than SCS and SCS+AA at first. At 15^{th} iteration, SCS+Neural has already reached a fixed-point residual of 0.001 while SCS takes about 50 iterations. However, when the number of iterations continues to increase, we can find a steady decrease in SCS. The residual of SCS+Neural, on the other side, sticks between 0.001 and 0.0001. This exactly matches the lower accuracy we mentioned above about SCS+Neural.

4.2 Other Application Results

Then, we apply *SCS+Neural* to more convex problems like Ridge (Hoerl and Kennard [1970]), SOCP problem in dense wireless cooperative networks(Shi *et al.* [2015]) and Compressed Sensing (Donoho [2006]) The results for SOCP problem in dense wireless cooperative networks are shown in Fig. 5 and Fig. 6. The results for Ridge are shown in Fig. 7 and Fig. 8.

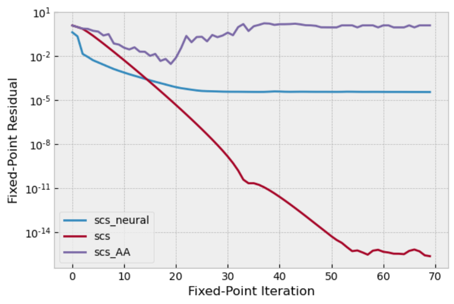

Figure 5: SCS+LSTM for SOCP

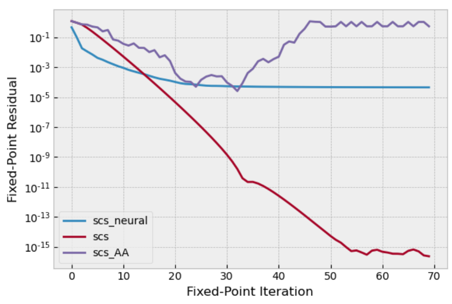

Figure 6: SCS+GRU for SOCP

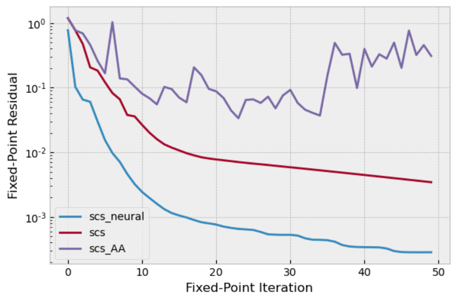

Figure 7: SCS+LSTM for Ridge

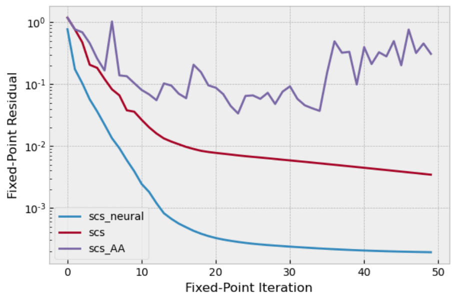

Figure 8: SCS+GRU for Ridge

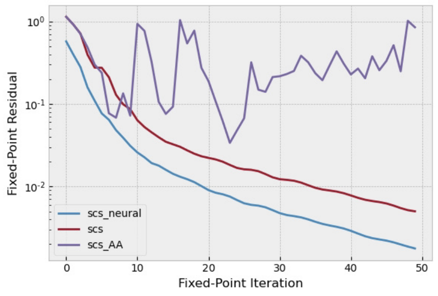

Figure 9: SCS+LSTM for Compressed Sensing

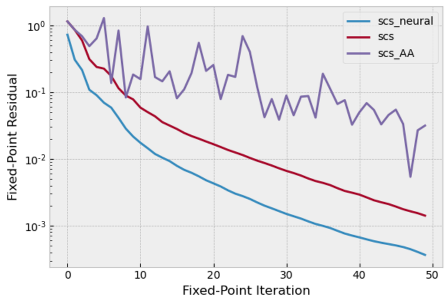

Figure 10: SCS+GRU for Compressed Sensing

The neural network does not work well on large-scale SOCP. Even if the size of RAUs and MUs increases slightly, the neural network might crash due to the huge demand of memory. Only in the case requiring less than 10 iterations, SCS+Neural performs better than SCS.

Still, SCS+Neural shows good performance when it comes to Ridge and Compressed Sensing problem. It only takes 15 iterations for the residual to reach less than 0.001. When we compare the structure of the neural network, the decrease of residual in GRU is more stable than LSTM. This phenomenon can also be found in other problems like Lasso.

5 Conclusions

In this report, we re-implemented the experiment in *Neural Fixed-Point Acceleration for Convex Optimization*, Venkataraman and Amos [2021] and applied it to other problems. We find that SCS+Neural performs better than SCS in many scenarios. We conducted a lot of numerical experiments in this project and tested many different hyper-parameters. In the future, we can continue to seek performance improvements by modifying the network structure or making changes to the SCS algorithm.

References

Alnur Ali, Eric Wong, and J Zico Kolter. A semismooth newton method for fast, generic convex programming. In *International Conference on Machine Learning*, pages 70–79. PMLR, 2017.

Donald G Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4):547–560, 1965.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

Kyri Baker. A learning-boosted quasi-newton method for ac optimal power flow. *arXiv preprint arXiv:2007.06074*, 2020.

Jonathan T Barron and Ben Poole. The fast bilateral solver. In *European Conference on Computer Vision*, pages 617–632. Springer, 2016.

Nicola Bastianello, Andrea Simonetto, and Emiliano Dall'Anese. Opreg-boost: Learning to accelerate online algorithms with operator regression. *arXiv preprint arXiv:2105.13271*, 2021.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Charles G Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of computation*, 19(92):577–593, 1965.

Enzo Busseti, Walaa M Moursi, and Stephen Boyd. Solution refinement at regular points of conic problems. *Computational Optimization and Applications*, 74(3):627–643, 2019.

Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):1–37, 2011.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *arXiv preprint arXiv:1704.01665*, 2017.

D.L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

Michael C Grant and Stephen P Boyd. Graph implementations for nonsmooth convex programs. In *Recent advances in learning and control*, pages 95–110. Springer, 2008.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pages 399–406, 2010.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021.

Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Luke Metz, Niru Maheswaranathan, Jonathon Shlens, Jascha Sohl-Dickstein, and Ekin D Cubuk. Using learned optimizers to make models robust to input noise. *arXiv preprint arXiv:1906.03367*, 2019.

Luke Metz, C Daniel Freeman, Niru Maheswaranathan, and Jascha Sohl-Dickstein. Training learned optimizers with randomly initialized learned optimizers. *arXiv preprint arXiv:2101.07367*, 2021.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Brendan O'donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Hypersolvers: Toward fast continuous-depth models. *arXiv preprint arXiv:2007.09601*, 2020.

Yuanming Shi, Jun Zhang, Brendan O'Donoghue, and Khaled B Letaief. Large-scale convex optimization for dense wireless cooperative networks. *IEEE Transactions on Signal Processing*, 63(18):4729–4743, 2015.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Shobha Venkataraman and Brandon Amos. Neural fixed-point acceleration for convex optimization. *arXiv preprint arXiv:2107.10254*, 2021.

Homer F Walker and Peng Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4):1715–1735, 2011.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International Conference on Machine Learning*, pages 3751–3760. PMLR, 2017.

Junzi Zhang, Brendan O'Donoghue, and Stephen Boyd. Globally convergent type-i anderson acceleration for nonsmooth fixed-point iterations. *SIAM Journal on Optimization*, 30(4):3170–3197, 2020.